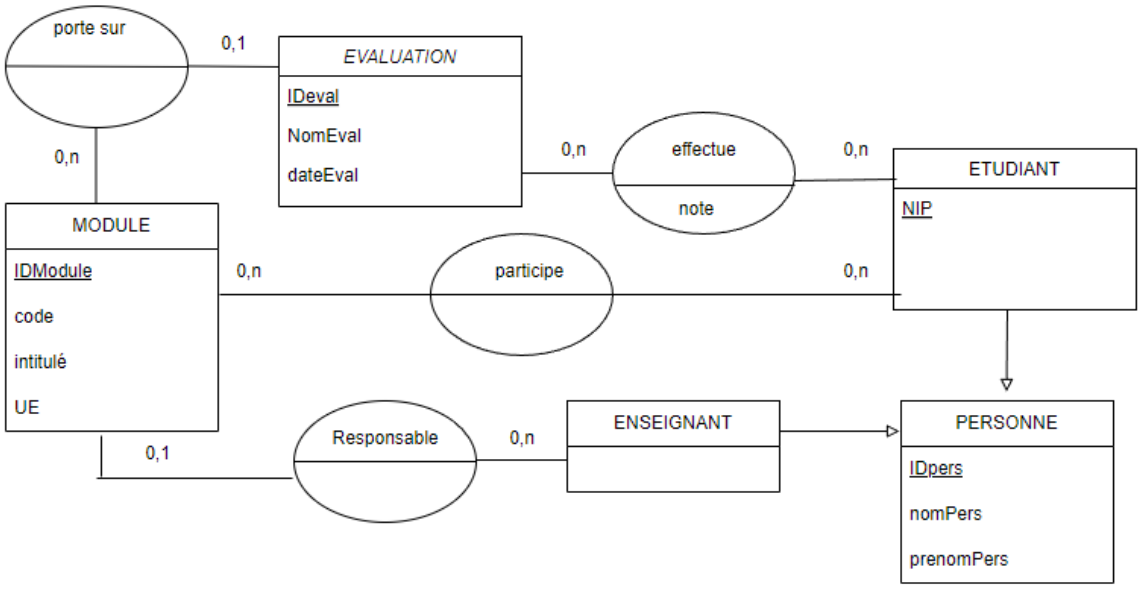
Baouchi Aboubakar Shango

Bases de données et langage SQL , Création d'une base de données

Sommaire

Modélisation et script de création « sans AGL » :

Modèle entités-associations :



Schema relationnel :

PERSONNE(id_personne, nom_personne , prenom_personne)

ETUDIANT(id_etudiant)

Id_etudiant fait référence à PERSONNE

ENSEIGNANT(id_enseignant)

Id_enseignant fait référence à PERSONNE

MODULE(id_module, intitulé, UE, code, id_enseignant)

Id_enseignant faut référence à ENSEIGNANT

EVALUATION(id_eval, nom_eval, date_eval, id_module)

Id_module fait référence à MODULE

aEffectue(idEval, idEtud, note)

idEval et idEtud font référence à EVALUATION et ETUDIANT

Participe(idModule,idEtudiant)

idModule et idEtudiant font référence à MODULE et ETUDIANT

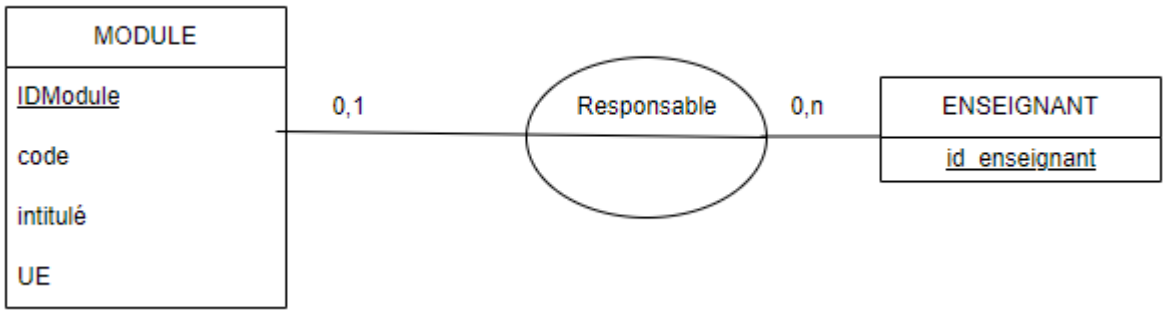
Script SQL de création des tables :

```
CREATE TABLE personne (
   id_personne INTEGER PRIMARY KEY, nom_personne VARCHAR,
prenom_personne VARCHAR);
```

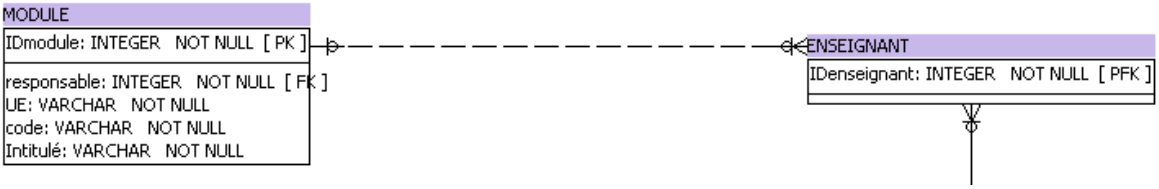
```
CREATE TABLE etudiant (
  id_etudiant INTEGER PRIMARY KEY REFERENCES personne(id_personne)
 );
CREATE TABLE enseignant (
  id_enseignant INTEGER PRIMARY KEYS REFERENCES
personne(id_personne)
 );

CREATE TABLE module (
  id_module INTEGER PRIMARY KEY,
  intitule VARCHAR,
  code VARCHAR,
  UE VARCHAR,
  Id_enseignant INTEGER REFERENCES enseignant(id_enseignant)
);
CREATE TABLE evaluation (
   id_eval SERIAL PRIMARY KEY,
  nom_eval VARCHAR,
  date_eval VARCHAR,
  Id_module INTEGER REFERENCES module(id_module)
 );

 CREATE TABLE participe(
  id_module INTEGER REFERENCES module(id_module),
  id_etudiant INTEGER REFERENCES etudiant(id_etudiant),
  PRIMARY KEY(id_Module, id_Etudiant)
);

CREATE TABLE estResponsable(
   id_module INTEGER REFERENCES module(id_module),
   id_enseignant INTEGER REFERENCES enseignant(id_enseignant),
  PRIMARY KEY(id_module, id_enseignant)
 );

CREATE TABLE aEffectue (
  id_eval INTEGER REFERENCES evaluation(id_eval),
  id_etud INTEGER REFERENCES etudiant(id_etudiant),
  PRIMARY KEY(id_Eval, id_Etud),
  note VARCHAR
);
```

Modélisation et script de création « avec AGL »

Illustration comparatives cours/AGL d'une association fonctionnelle



Sans AGL

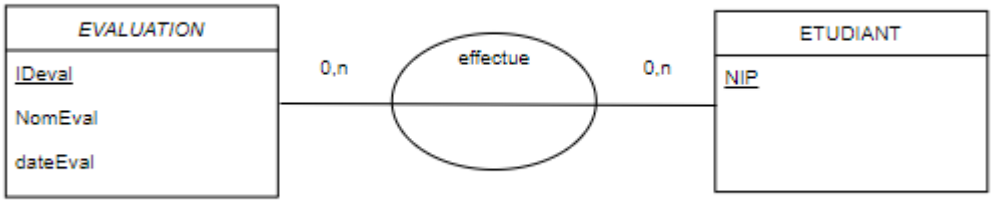


Avec AGL

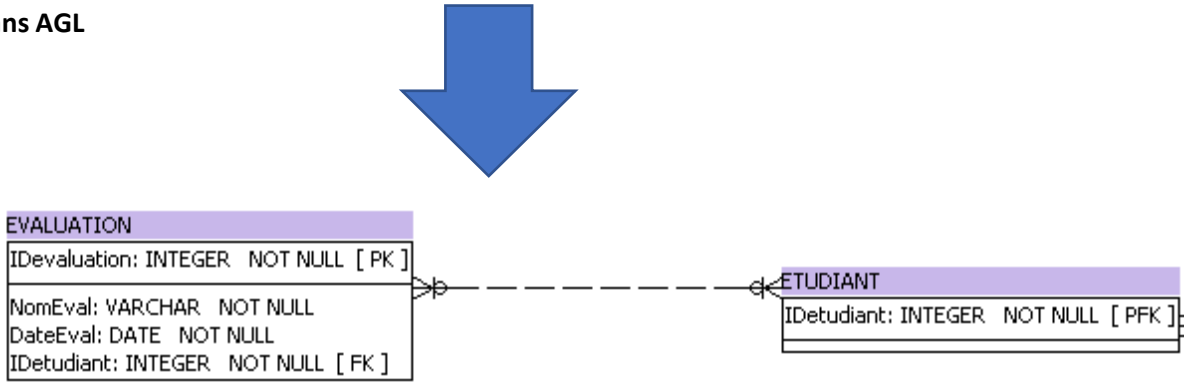
Les deux schémas ci-dessus représente la même association fonctionnelle d'abord sans AGL puis avec. Elle est inspirée du sujet de la SAE mais n'est pas exactement le même . C'est bien une association fonctionnelle car il y a une cardinalité maximale de 1 du côté de MODULE. En effet un module ne peut avoir qu'un responsable au maximum. Commençons la comparaison en remarquant que le type association n'est pas représenté par l'AGL mais une clé étrangère faisant référence à ENSEIGNANT vient directement dans les clés de MODULE.

On peut clairement le voir avec la surbrillance lorsque l'on clique sur le lien. Côté AGL, les clés priamaires sont séparés des clés par une barre horizontale. Cela remplace le soulignement des clés primaires sans AGL. Concernant les cardinalités , la version AGL ne posséde pas de chiffre mais des symboles . Ici le côté droit posséde tun cercle et trois barres qui correspondent respectivement à 0 et n. De l'autre côté nous avons un cercle et une seule barre correspondant à 0 et 1.

Illustration comparative d'une association maillée.



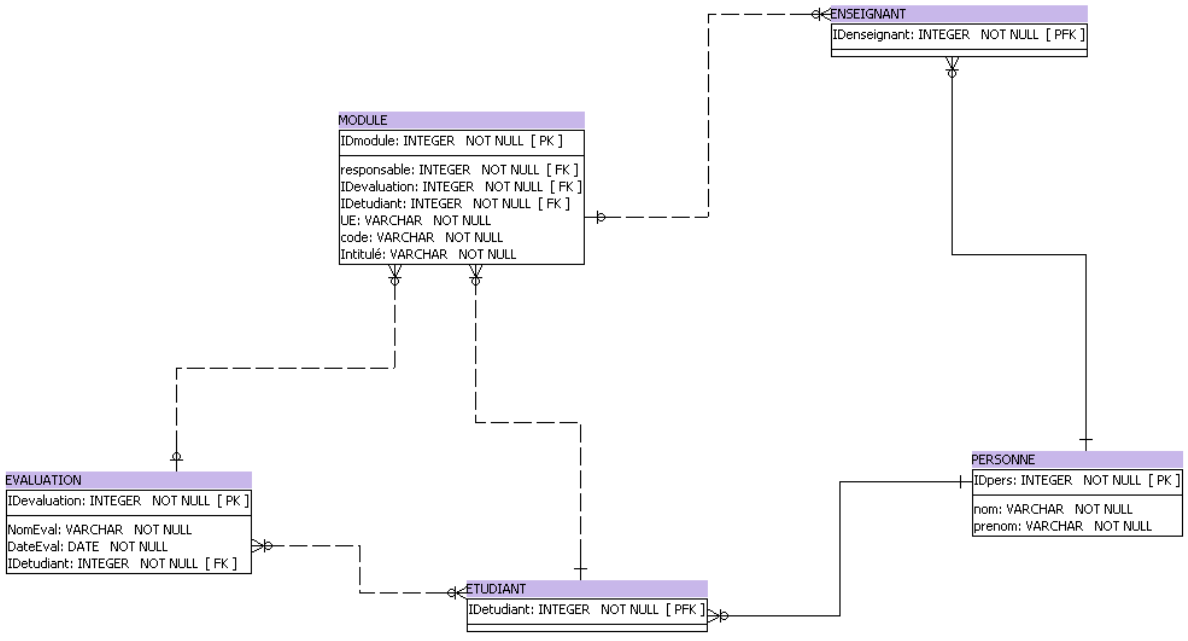
Sans AGL



Avec AGL

On remarque qu'il n'y a pas beaucoup de changement changement entre ces deux représentation si ce n'est les différences liées au format de l'AGL : pas de type association représenté , pas soulignement pour les clés primaires , symbole à la place de chiffre pour les cardinalités. Or, nous avons déjà présenté ces spécificités plus haut.

Modèle entités-associations réalisé avec l'AGL



Script SQL de création de table généré par AGL

```
CREATE TABLE public.PERSONNE (
                IDpers INTEGER NOT NULL,
                nom VARCHAR NOT NULL,
                prenom VARCHAR NOT NULL,
                CONSTRAINT idpers PRIMARY KEY (IDpers)
);


CREATE TABLE public.ENSEIGNANT (
                IDenseignant INTEGER NOT NULL,
                CONSTRAINT idenseignant PRIMARY KEY (IDenseignant)
);


CREATE TABLE public.ETUDIANT (
                IDetudiant INTEGER NOT NULL,
                CONSTRAINT nip PRIMARY KEY (IDetudiant)
);


CREATE TABLE public.EVALUATION (
                IDevaluation INTEGER NOT NULL,
                NomEval VARCHAR NOT NULL,
                DateEval DATE NOT NULL,
                IDetudiant INTEGER NOT NULL,
                CONSTRAINT idevaluation PRIMARY KEY (IDevaluation)
);


CREATE TABLE public.MODULE (
                IDmodule INTEGER NOT NULL,
                responsable INTEGER NOT NULL,
```

```
                IDevaluation INTEGER NOT NULL,
                IDetudiant INTEGER NOT NULL,
                UE VARCHAR NOT NULL,
                code VARCHAR NOT NULL,
                Intitul VARCHAR NOT NULL,
                CONSTRAINT idmodule PRIMARY KEY (IDmodule)
);


ALTER TABLE public.ETUDIANT ADD CONSTRAINT personne_etudiant_fk
FOREIGN KEY (IDetudiant)
REFERENCES public.PERSONNE (IDpers)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE public.ENSEIGNANT ADD CONSTRAINT personne_enseignant_fk
FOREIGN KEY (IDenseignant)
REFERENCES public.PERSONNE (IDpers)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE public.MODULE ADD CONSTRAINT enseignant_module_fk
FOREIGN KEY (responsable)
REFERENCES public.ENSEIGNANT (IDenseignant)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE public.EVALUATION ADD CONSTRAINT etudiant_evaluation_fk
FOREIGN KEY (IDetudiant)
REFERENCES public.ETUDIANT (IDetudiant)
```

```
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE public.MODULE ADD CONSTRAINT etudiant_module_fk
FOREIGN KEY (IDetudiant)
REFERENCES public.ETUDIANT (IDetudiant)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE public.MODULE ADD CONSTRAINT evaluation_module_fk
FOREIGN KEY (IDevaluation)
REFERENCES public.EVALUATION (IDevaluation)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```

Comparaison entre script manuelle et script généré par AGL

La création des tables et très similaire malgrès que le script automatique posséde plus d'information. Par exemple l'AGL va automatiquement donné la condition NOT NULL aux colonnes. On remarque également que le script automatique se fait en plusieurs temps. Il crée dans un premier temps les tables puis les modifie avec ALTER TABLE et va préciser la deferabilité.

PEUPLEMENT DES TABLES ET REQUETES

Description commentée des étapes du script de peuplement

Premièrement nous allons créer une table data dans laquelle nous allonsmettre l'entiereté du fichier csv récupéré. Il faut donc mettre toutes les colonnes du fichier.

```
CREATE TABLE data (
  id_enseignant INTEGER,
   nom_enseignant VARCHAR,
   prenom_enseignant VARCHAR,
  id_module INTEGER,
   code VARCHAR,
   ue VARCHAR,
   intitule_module VARCHAR,
   nom_evaluation VARCHAR,
   date_evaluation VARCHAR,
   note VARCHAR,
   id_etudiant INTEGER,
   nom_etudiant VARCHAR,
   prenom_etudiant VARCHAR

);
```

On va ensuite copier le contenu du fichier dans notre nouvel table .

\copy data FROM '/tmp/data.csv' WITH(FORMAT CSV,HEADER ,DELIMITER' ;')

Le WITH va ici nous permettre de lire le contenu du fichier et de ne pas prendre en compte les premières lignes inutiles .

Maintenant que l'on a retranscrit le contenu du fichier dans un table, on va remplir les tables que l'on a créer avec notre scipt manuel grâce des requêtes. On va utiliser les commande INSERT INTO (pour remplir les cibles) et SELECT(séléctionner ce que l'on va copier).

```
 INSERT INTO
personne(id_personne, nom_personne, prenom_personne)
SELECT
  DISTINCT id_enseignant, nom_enseignant, prenom_enseignant
FROM
  data;

INSERT INTO
 etudiant(id_etudiant)
 SELECT
   DISTINCT id_etudiant
FROM
   data;

 INSERT INTO
 enseignant(id_enseignant)
 SELECT
  DISTINCT id_enseignant
 FROM
   data;

INSERT INTO
 module(id_module, intitule, code, UE)
 SELECT
   DISTINCT id_module, intitule_module, code ,ue
 FROM data;
```

```
 INSERT INTO
 evaluation(nom_eval, date_eval, id_module)
 SELECT
   DISTINCT nom_evaluation, date_evaluation, id_module
 FROM data;

 INSERT INTO
 participe(id_module, id_etudiant)
 SELECT
  DISTINCT id_module, id_etudiant
 FROM data;

INSERT INTO
estResponsable(id_module, id_enseignant)
SELECT
  DISTINCT id_module, id_enseignant
FROM data ;

INSERT INTO
aEffectue(id_eval, id_etudiant)
SELECT
  DISTINCT id_evaluation, id_etudiant
FROM data ;
```

On a bien rempli les tables que l'on a créé grâce aux requêtes. On peut si on le souhaite supprimer la table data.

Requêtes intéressantes

Commençons par une requête qui va permettre d'afficher un classement des élèves selon leurs moyennes.

```
SELECT
  nom_personne,
  prenom_personne,
  AVG(aeffectuer.note)
FROM
  aeffectue
  JOIN personne ON id_etud = id_personne
GROUP BY
  nom_personne,
  prenom_personne, note
ORDER BY note DESC;
```

Pour cette requête, on effectue une jointure entre la table PERSONNE et la table AEFFECTUE en faisant une projection avec les colonnes nom_personne, prenom_personne et AVG des notes qui va permettre d'avoir la moyenne des notes de chaque étudiant grâce au GROUP BY. On ajoute un order by note DESC pour avoir un classement .

Nous allons maintenant faire une requête qui va permettre de donner le nombre de module dont est responsable un enseignant.

```
SELECT
  nom_personne,
  prenom_personne,
  COUNT(DISTINCT id_module)
FROM
  estresponsable
  JOIN personne ON id_enseignant = id_personne
GROUP BY
  nom_personne,
  prenom_personne, id_module
ORDER BY COUNT(DISTINCT id_module) ;
```

Pour cette requête, on effectue une jointure entre la table PERSONNE et la table ESTRESPONSABLE en faisant une projection avec les colonnes nom_personne, prenom_personne et id_module qui va permettre de savoir de combien de module est responsable un enseignant .